A Distributed System for Supporting Spatio-temporal Analysis on Large-scale Camera Networks

Kirak Hong*, Marco Voelz†, Venu Govindaraju‡, Bharat Jayaraman‡, and Umakishore Ramachandran*
*Georgia Institute of Technology
{khong9, rama}@cc.gatech.edu
†University of Stuttgart
marco.voelz@ipvs.uni-stuttgart.de
†SUNY Buffalo
{govind, bharat}@buffalo.edu

***Abstract*—Cameras are becoming ubiquitous. Technological advances and the low cost of such sensors enable deployment of large-scale camera networks in metropolises such as London and New York. Applications such as video-base surveillance and emergency response that exploit such camera networks are continuous, data intensive, and dynamic in terms of resource requirements. Common anomalies in such application spaces include authorized personnel moving into unauthorized spaces and checking the movement of suspicious individuals as they move through the spaces. High level goal in such applications include catching such anomalies in real time and reducing collateral damage. A well-known technique for meeting this high level goal is spatiotemporal analysis. This is an inferencing technique employed by domain experts (e.g., vision researchers) to answer queries such as show the track of person A in the last 30 minutes. Performing spatio-temporal analysis in real-time for a large-scale camera network is challenging. It involves continuously capturing images from distributed cameras, analyzing the images to detect and track objects of interest in the field of view of the cameras, generating an event by comparing the signature of a detected object against a database of known signatures, and maintaining a state transition table indexed by time that shows the spatio-temporal evolution of people movement through the distributed spaces. In this paper, we propose a distributed system architecture to address these challenges. We make the following contributions: (a) present the design choices for real-time spatio-temporal analysis with a view to supporting scalability (in terms of number of cameras, event rate, and known targets), (b) develop heuristics for pruning the event generation phase of spatio-temporal analysis, and (c) implement and evaluate the different design choices in a distributed system to show the scalability of our distributed system architecture.**

I. Introduction

As sensors for recognizing humans, such as cameras, voice recognition sensors, and RFID readers, are becoming more capable and widely deployed, new application scenarios arise, requiring an automated processing of the continuous stream data to identify and track human beings in real-time. Scenarios in this field include airport security, emergency response and assisted living, all requiring real time detection of unusual situations, called *anomalies*. Different from techniques such as RFID badges, cameras allow for an unobtrusive way of identifying people's whereabouts, making them the primary source of information in many of these scenarios.

Take an airport scenario as an example: Amsterdam's Schiphol airport currently has 1,000 cameras in place and plans to increase that number to between 3,000 and 4,000 over the next few years [1]. In an airport, a common security violation is that an individual enters into a restricted area without permission. If such a situation happens, the individual should be reported to an airport security team in real time, preventing potential threats to the airport. Similarly, any individuals who checked in their baggage but did not board their airplane or unattended baggages are other examples of anomalous situations in an airport.

The high level goal in such applications, often referred to as *situation awareness* applications [2], is catching anomalies in real time and reducing collateral damage. To achieve this, there is a well-known technique called *spatio-temporal analysis*, enabling an application to answer queries such as "Where is person A?", "When and where did person A leave zone X?", "When and where did person A and person B meet for the last time?".

Applications providing the means to answer these queries usually employ distributed cameras and sensors of other modalities (such as audio and biometrics) to detect people in the observed system. These live sensor streams are used to make an estimation about the identity of the detected people, comparing the data to a set of well-known identities. These estimates generated throughout the system are gathered and regularly consolidated to create a global view of the observed area, e.g., by recording the most likely whereabouts of each person known to the system at a certain point of time. The current global state and possibly a history of former states enable the system to answer queries such as stated above.

Recently, Menon et al. [3] showed the feasibility of spatio-temporal analysis using this concept by maintaining the global state in a transition table similar to hidden markov models. The table represents the probabilities of each person known to the system being in each of the observed locations. Events,

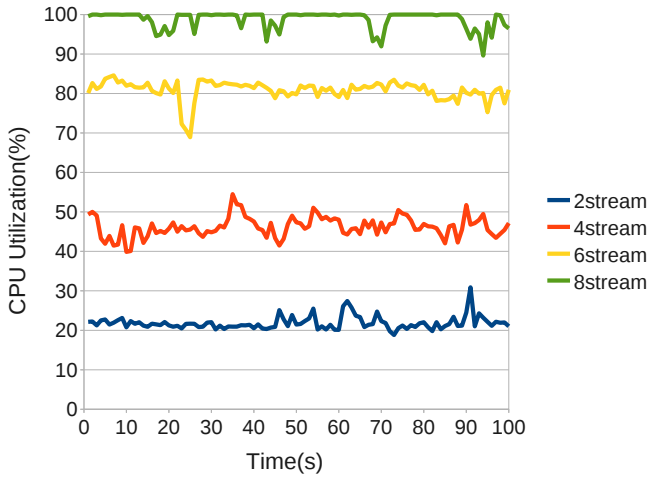

Fig. 1. CPU utilization for different number of streams in a centralized node: Cameras produce 640x480 videos at 10 fps. Target detection and tracking happen on a 2.93GHz Intel i7 870 processor.

which indicate that a person has entered an observed area, trigger a transition from the current state to the next. Just as the global state, events are represented by probabilities rather than exact knowledge, because algorithms for signature detection and comparison are inherently inaccurate.

There are several steps involved in running spatio-temporal analysis on camera networks. When a person appears in the field of view of a camera, the person is detected as a target and is then tracked within the camera view. While the person is being tracked, multiple features such as face images are collected in order to generate a representative signature of that target e.g., by using the eigenface algorithm [4]). When sufficient features have been collected, a representative signature for the target is generated. The signature is then compared against known signatures in a database using computer vision techniques such as face recognition algorithms. The output of such recognition algorithms is an *event*, which is then used by the system to update its state, reflecting an up-to-date location of targets.

As image processing and interpretation tasks migrate from a manual to a computer-automated model, questions of system scalability and efficient resource management will arise and must be addressed. In large settings such as airports or urban environments, processing the data streaming continuously from multiple video cameras is a computationally intensive task. Moreover, given the goals of real-time spatio-temporal analysis, images must be processed in real time in order to provide the timeliness required by modern security practice. Questions of system scalability go beyond video analytics, and fall squarely in the purview of distributed systems research.

We identify the challenges in providing spatio-temporal analysis on large-scale camera networks. First, target detection and tracking algorithms are generally computationally intensive. Figure 1 shows the CPU utilization of a single 2.93GHz Intel Core i7 870 machine when performing target detection and tracking on 640x480 live video streams at 10 frames per second; as can be seen, all of the 8 logical cores are fully utilized with 8 streams. Second, signature comparison for target recognition is another computationally intensive task. As Table I indicates, even the simplest algorithm takes 5 milliseconds to compare one face to three thousand faces. Algorithms considering additional features could be even more demanding of the CPU time. This would become a bottleneck in a large-scale camera network since potentially a number of signatures would be detected simultaneously, each of which would require comparison to a known set of signatures. Third, writing a large scale distributed program for spatio-temporal analysis is difficult for a domain expert (e.g., computer vision researcher), due to the complexities involved in distributed processing.

N of Faces	PCA	LDA	Bayesian	EBGM
600	1	4	1954	5056
1200	1	9	4072	20250
1800	3	14	5533	30282
2400	4	19	7867	80695
3000	5	24	9806	100892

TABLE I
EXECUTION TIME FOR COMPARING ONE FACE TO DIFFERENT NUMBER OF FACES: DIFFERENT ALGORITHMS ARE EVALUATED USING CSU FACE IDENTIFICATION EVALUATION SYSTEM.

The problem being addressed in this paper can be stated as follows: How to develop a distributed system for real-time spatio-temporal analysis on a large scale camera network?

Specifically, we make the following contributions:

- We provide a distributed framework for real-time spatio-temporal analysis that can be used by domain experts to "plug and play" their algorithms without worrying about the details of the distributed computing issues.
- We present design choices for real-time spatio-temporal analysis with a view to supporting scalability in terms of number of cameras, event rate, and known targets.
- We propose heuristics to reduce the computational cost of event generation by selectively comparing a signature to nearby signatures based on physical adjacency.
- We implement the design choices in a distributed system, and evaluate their efficacy for supporting scalability of real-time spatio-temporal analysis.

Section II explores the design space of spatio-temporal analysis on large scale camera networks. Section III gives implementation details of the selected design choices. Section IV discusses results of our scalability studies, and Section VI presents concluding remarks.

II. System Architecture

In this section, we give an overview of our approach to provide efficient spatio-temporal analysis in a large-scale camera network. First, we detail the processing steps involved in spatio-temporal analysis. Second, we discuss the design space of distributing the individual steps of spatio-temporal analysis and choose two promising designs. Third, we introduce the building blocks for realizing the processing steps according to the selected designs. Last, we present heuristics to reduce the

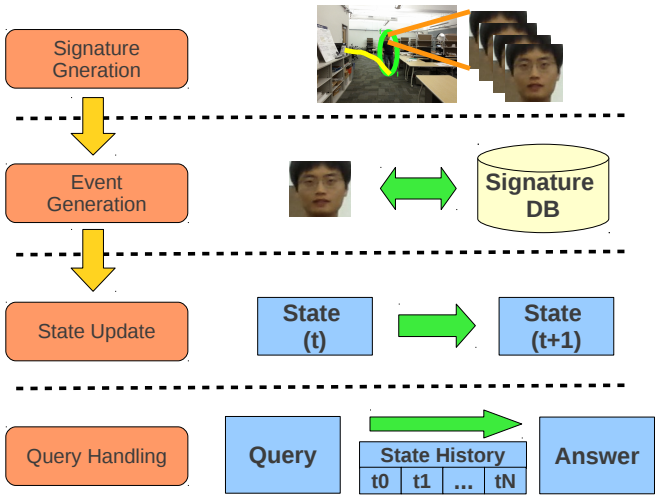

Fig. 2. Application structure for spatio temporal analysis: This is a general structure of spatio-temporal analysis on camera network that answers target locations at given time using camera streams.

computational cost of event generation by selective signature comparison.

A. Spatio-temporal Analysis

In general, spatio-temporal analysis enables an application to answer queries referring to locality- and time-dependent information about different targets. Common examples of spatio-temporal queries include:

"Where was person A at time T?"
"When did person B leave zone X?"
"When and where did person A and B meet for the last time?"
"Who moved from zone X to zone Y between time T1 to T2?"

To answer these queries, an application has to maintain its state which represents each individual's location at a certain point of time. Figure 2 shows a general structure of spatio-temporal analysis on a camera network involving four steps: Signature generation for a new target, event generation, state update, and query handling.

Signature generation involves various video analytics such as target detection, image capture, and signature extraction. When a person moves into a cameras field of view, that person will be detected as a new target. Once detected, the target is followed in the camera view, face images are captured from the individual frames, and a signature for each face image is constructed. From the multiple signatures, one representative signature is generated during this phase. For example, a summarization of face images called eigenface [4] can be built from the collected face images. Alternatively, the system may choose to report each detected signature rather than a representative signature. However, this will result in poor accuracy due to the increased number of events based on less accurate signatures.

Event generation involves comparing the target signature to known signatures in a database. The goal is to come up with an estimate about the identity of a person who was detected in a camera observed area. Depending on the application needs, different algorithms can be used in this step, each being potentially computing intensive. For example, the estimated probability that a detected person is not known to the system yet influences the type of signature comparison algorithm that should be chosen by application developers [5], [6].

State update maintains an application-specific state based on an event generated. The goal is to reflect the information provided by an event, e.g., that Person A was seen in Zone 2 with a probability of 0.75, in the global state. A state of an application includes its knowledge about each target's location at a given time. A new event causes an update from one state to another, as it reports a new information about target locations. Different implementations for maintaining system state are possible, depending on the application's information needs. For example, Menon et al. [3] proposed a state transition system similar to hidden markov models. Each state is represented as a table indicating the probabilities of each person being in a specific zone. In their work, events are given by a simulator using predefined target movements, thereby abstracting the first two steps of signature and event generation. An event is a vector, where each component is a similarity between a detected signature and one of the known signatures in a database. The state transition function uses an event to update all entries in the state transition table, resulting in new probabilities of peoples' whereabouts. A sequence of all states is stored persistently, thereby maintaining trajectories of the people in the system.

Our goal is to provide a framework for efficient spatio-temporal analysis according to these processing steps in a large-scale camera network. The framework is independent from the specific algorithms used to realize the specific steps and treats them as black-boxes, such that domain experts can plug-in the functionality suited best for their application needs. The next section explores the design space and selects two promising approaches to tackle the main performance challenges by distributing individual processing steps.

B. Exploration of the Design Space

In order to support distributed real-time spatio-temporal analysis, we need to make assumptions for sensing and compute infrastructures. For sensing infrastructure, we assume that there exist distributed networked smart cameras as well as other sensing modalities to improve accuracy and reduce false positives and negatives. The smart cameras can perform video analytics locally, which reduces network infrastructure overhead and therefore improves scalability. We also assume a compute infrastructure that can elastically increase/decrease computational resources based on need, such as a cloud. The compute infrastructure will provide the workhorses to deal with the dynamic workload of spatio-temporal analysis on a large scale camera network.

Figure 2 shows the data flow and relations between the processing steps described in the previous section. We did not consider query handling for the system design since it is not

	Signature Generation	Event Generation	State Update
I	Distributed	Centralized	Centralized
II	Distributed	Distributed	Centralized
III	Distributed	Distributed	Distributed

TABLE II
DESIGN CHOICES FOR DISTRIBUTED SPATIO-TEMPORAL ANALYSIS

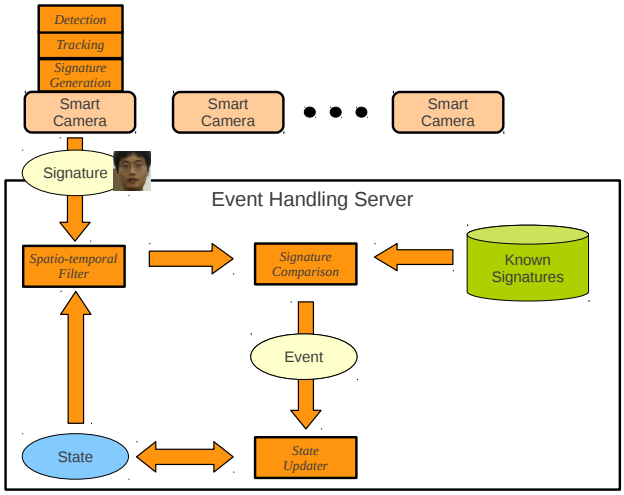

Fig. 3. Centralized event generation: Candidate signatures decided by a spatio-temporal filter are compared against a newly detected signature at a centralized node.

in the critical path of real-time event processing. Providing distributed computing support for efficient query processing is our future work.

To explore design spaces for real-time spatio-temporal analysis, we consider each step in Figure 2 to be either centralized or distributed. Table II shows the three design choices we consider for distributed spatio-temporal analysis.

The first design runs video analytics for signature generation on distributed smart cameras. However, event generation and state update are done in a centralized fashion (cf. Figure 3). In this case, signature comparison between a detected signature and known signatures happens at a single node for the targets detected (potentially simultaneously) by the system. Thus the event generation step could become a bottleneck when there are a large number of signatures detected simultaneously from distributed cameras and each signature needs to be compared against a large number of known signatures. Using complex algorithms for signature comparison will exacerbate the performance bottleneck.

The second design further increases scalability by distributing the workload for event generation. This essentially distributes signature comparison over multiple distributed workers in order to process large number of signature comparisons in real time. For example, if there are 1,000 signatures in a database, 10 distributed workers can speed up the event generation process by performing 100 comparisons per worker. Once signature comparisons are done by distributed workers, the results are combined to generate an event. The event is consumed by a centralized state update module which makes a state transition from the current state to the next state. The overhead of state update depends on the complexity of the state transition algorithm. In general, however, the computational cost for state update is much smaller than previous steps since it works on highly abstract (and already processed) data such as probabilities rather than raw videos or signatures. For instance, the execution time of the state transition function used in our implementation for a single event is in the order of microseconds.

The third design additionally distributes the state update over multiple nodes. However, many applications using spatio-temporal analysis require a globally consistent state. Typically, the algorithms written by domain experts for spatio-temporal analysis (e.g., the one by Menon et al. [3]) are centralized. To fully distribute such an algorithm would require significant amount of engineering. For example, with reference to Figure 4, to ensure that the temporal evolution of the replicated state transition table is globally consistent, application of events to the table has to be orchestrated using sophisticated techniques (e.g., Lamport clock [7]) to ensure a total order. Further, such a fully distributed implementation would complicate query processing if all the replicated copies are not kept consistent.

Considering the fact that an update to the state transition table is not as computationally demanding as the other two steps, and to ensure that our distributed solution presents an easy migration path for domain experts to use, we restrict our design choices to I and II in Table II.

C. Distributing Spatio-temporal Analysis

The design choices presented in the previous section refer to distributing the various processing steps of spatio-temporal analysis. In this section, we outline techniques for distributing the *signature generation* and the *event generation* steps, while remaining oblivious to the specific algorithms used by the domain experts.

Signature generation: To distribute signature generation, we make use of the capabilities of smart cameras and use a programming model called Target Container (TC) [8]. The key insight in the TC programming model is to elevate *target* as a first class entity both from the perspective of the programmer and from the perspective of resource allocation by the execution environment. Consequently, all application level vision tasks become target-centric, which is more natural from the point of view of the domain expert.

Using TC, a domain expert is required to provide *handlers* that implement algorithmic details of video analytics, such as target detection and tracking, while the distributed system issues including facilitating the dynamic instantiation of these handlers, communication and synchronization for the data structures shared by these handlers are handled by the framework. In Target Container programming model, there are four different handlers: detector, tracker, equality checker and merger.

The role of the detector is to analyze each camera image it receives to detect any new target in a camera's field of view. The detector creates a *target container* for each new target it recognizes. Once a target is detected, the system spawns a tracker and associates it with this camera to track the target

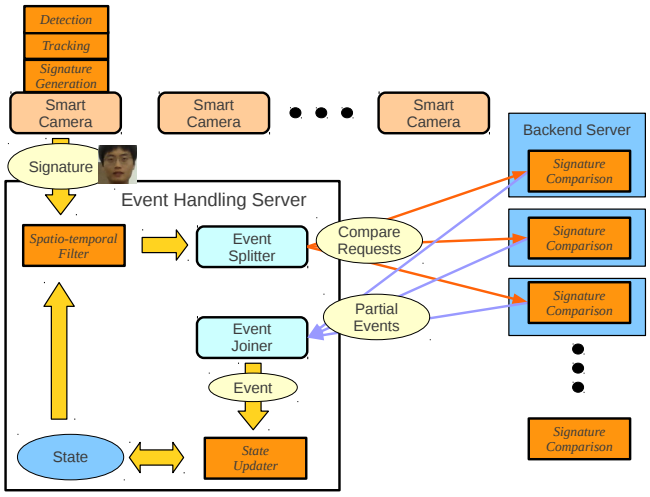

Fig. 4. Distributed event generation: The domain expert gives a filter (ST filter) that our infrastructure uses to generate the signature set against which the incoming signature needs to be compared. Backend computing resources are utilized to perform the signature comparison in parallel. The results from the backend are combined to generate an *event* that is then returned to the *state update* handler provided by the domain expert.

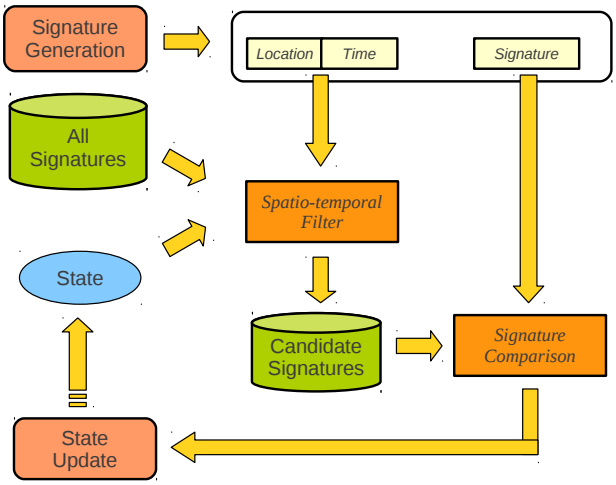

Fig. 5. The spatio-temporal filter is an application-specific handler provided by the domain expert for selecting a subset of signatures against which the detected signature needs to be compared.

A tracker can notify the TC system that this tracker need not be scheduled anymore when its target is leaving the field of view of the camera that it is associated with.

Detector and tracker play key roles in spatio-temporal analysis on a camera network since they can tell when a target enters a single camera's field of view and when the target leaves. To implement a spatio-temporal analysis engine, the domain expert would extend the tracker functionality to generate a *target signature* with *time* and *location* (cf. Figure 4)[1].

Event generation: To distribute event generation, we make use of the backend computing resources (either a dedicated cluster or elastic computing resources such as a cloud) to distribute the signature comparison that leads to event generation.

Figure 4 shows the implementation of spatio-temporal analysis with distributed event generation. In addition to the detector and tracker, the domain expert provides two additional handlers: *ST filter* and *State Updater*. The former handler selects a sigature subset from the signature database that needs to be compared against the incoming signature generated by the tracker. *Event Splitter* and *Event Joiner* are part of our distributed system infrastructure. Once a signature is generated using the detector and tracker, it is sent to the *Event Handling Server*, which is a master node in our distributed architecture. The signature is handed to the *Event Splitter*, which uses the signature subset from the ST filter to distribute the signature comparison to backend computing resources. Each backend server, is assigned a non-overlapping sublist of the signatures to be compared against the detected signature. The database of signatures is assumed to be known *a priori*, and hence replicated on all the backend servers. The comparison operations carried by the backend servers are embarrassingly parallel, requiring no communication among the servers. Once completed, each backend server sends back its result to the master node. The Event Joiner combines these partial results to generate an *event*, which is then passed on to the State Updater handler provided by the domain expert. The distributed architecture uses ActiveMQ [9] for communication among the master and backend servers.

[1]In the TC framework, the domain expert may provide additional handlers for *equality checking* of targets appearing in the FOV of multiple cameras, and *merging* the associated TCs to create a single TC containing the trackers for the multiple cameras. However, for the spatio-temporal analysis engine being described in this paper these additional handlers are superflous since the the event generation and state update steps of the spatio-temporal analysis subsumes their functionality.

D. Selective Event Generation

Event generation in spatio-temporal analysis often becomes a bottleneck in a large scale scenario. This is because there are potentially large number of signatures generated, which causes huge number of comparisons for event generation. Plus, the large number of known signatures in the database as well as the computationally intensive nature of the signature comparison algorithms impose additional overheads for event generation.

One generic optimization technique for event generation is to compare a detected signature to a smaller subset of the signatures in the database rather than the entire set of known signatures. Figure 5 shows our framework for such a selective event generation. When a signature is generated, it is tagged with its detected location and time. The time and location information are then used by the *spatio-temporal filter* supplied by the domain expert, which provides candidate signatures for comparison based on location and time information of a detected signature. As the spatio-temporal filter is an application-specific code module written by the domain expert, it embodies state of the application and other knowledge such as the physical layout of the sensors and the building floor plans. For the purposes of the distributed architecture presented in this paper, we treat this filter as a

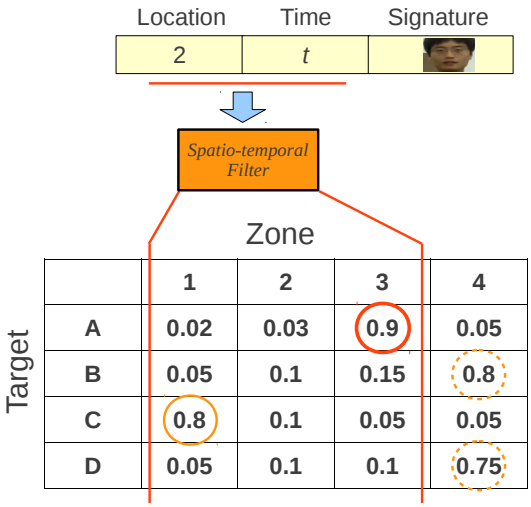

Fig. 6. An example of how a spatio-temporal filter may be designed to prune the search space of signatures to be compared using the time and space attributes of the generated signature

black box.

Figure 6 shows a concrete example of the spatio-temporal filter based on the state transition table used in [3]. For the sake of exposition as to the intuition that goes into designing such a filter consider the state transition table in Figure 6, that represents the current state of the spatio-temporal analysis using the technique discussed by Menon et al. [3] In the example, a face is detected in Zone 2. A signature is generated and presented to the system. An application-specific filter would limit the search space to zones 1, 2, and 3, assuming that the adjacency in the state transition table (shown in Figure 6) implies physical adjacency. The reasoning behind pruning the search space thusly may be based on attributes of the physical space (e.g., distance between the zones, any blind spots where there may not be camera coverage between zones, etc.), and knowledge about the expected velocity of objects moving in the physical space. Further, referring to the table in Figure 6, the current state indicates that the probabilities associated with targets A and C in zones 3 and 1 are high, while those for B and D are low. Therefore, the spatio-temporal filter would choose A and C as the signature subset to compare the detected signature and may choose not to include B and D in that subset. Assume that in fact it was target A that moved from Zone 3 to Zone 2. The comparison would result in the system generating an event vector in which the detected signature will have a high probability that it matches A (and not C). This event vector would then be applied to the current state transition table resulting in the evolution of the table to a new state wherein Zone 2 will show a high probability for A being present in that zone.

Note that this framework is generic since an application can choose to use all target signatures if it does not want to conduct spatio-temporal filtering. However, the application developer should remember that the spatio-temporal filter has direct implications on both accuracy and performance.

E. Putting it all Together

The domain expert interacts with the proposed distributed architecture by providing the following handlers: *detector*, *tracker*, *signature comparison*, *state updater*, and *spatio-temporal filter*. The domain expert also provides the system with the database of signatures. The system uses these handlers as described in the earlier subsections to orchestrate the temporal evolution of the state transition table as people move through the space and are detected by the cameras.

III. Implementation

Centralized Event Generation: The video analytics for signature generation are performed on smart cameras using the the detector and tracker handlers provided by the domain expert. Once signatures are generated, they are sent to a centralized node called *Event Handling Server* with Intel(R) Core(TM)2 Quad CPU Q8200@2.33GHz. The event processing server performs all computations necessary for event generation as well as state update using its local system resources. When it receives a detected target with its associated signature, it performs spatio-temporal filtering based on the the attributes (space and time) of the detected signature. The spatio-temporal filter takes time and location associated with a signature, and provides a list of candidate signatures as output for signature comparison. Once all comparisons between the detected signature and candidate target signatures are done, an event is generated. This event is used by the state update module to generate the next temporal evolution of the state transition table.

Distributed Event Generation: As shown in Figure 4, we utilize a set of servers for distributed signature comparison. Similar to the first implementation with centralized event generation, we use smart smart cameras running video analytics including detector and tracker for signature generation.

To provision the servers needed for distributed signature comparison, we created distributed nodes using Amazon Elastic Compute Cloud (EC2) [10]. Each of the signature comparison module is running on a *High-CPU Extra Large Instance*, which is one of the highest class of instance types provided by Amazon EC2. Each instance has 7 gigabytes of memory and 20 *EC2 Compute Units* that is a virtualized computing capacity. Even though the instances are virtualized nodes, we observed consistent execution times for the same amount of workload. Also, the network bandwidth and latency between our event handling server (outside of the cloud) and signature comparison nodes (inside of the cloud) were reasonable to utilize a large number of cloud instances, as shown in our experimental results. For communication between the event handling server and distributed signature comparison modules, we used asynchronous messages of ActiveMQ [9].

Algorithmic Details for Spatio-temporal Analysis: To have a complete prototype implementation of the spatio-temporal analysis engine on top of our distributed system we implemented the handlers that we described in Section II-E. The detector handler implements a face detection algorithm

from the OpenCV [11] library. The tracker handler implements a color-based tracking algorithm [12]. When a target is detected in the field of view of a camera, the detector creates a new tracker that follows the target within the camera view while collecting target signatures. We use a face recognition algorithm from CSU Face Recognition Evaluation System [13] for signature generation. The face detection algorithm is applied to each target bounding box that is calculated by the target tracking algorithm. If a face is found from a target bounding box, the face recognition algorithm is used to generate a face feature (a signature).

For event generation, we modified CSU Face Recognition Evaluation System [13] to store preprocessed features of face images. The original system provides implementation of various face recognition algorithms with evaluation code which compares a given face image to other face images. Since the intention of the original system is to evaluate different face recognition algorithms, it loads and processes benchmark face images every time it runs. We modified the original system to suit our needs for signature comparison. We preprocess all the benchmark face images and store face features in each signature comparison server. When a signature comparison request message is arrives, the system compares the face feature included in the message against to number of face features that are already preprocessed. For signature comparison, we used *Bayesian face recognition algorithm* from the CSU Face Recognition Evaluation System. Face images are obtained from the Facial Recognition Technology (FERET) Database [14] which includes about 3,000 faces[2].

After an event is generated, the event is used by a state update module for updating the application state. Specifically, we use the spatio-temporal analysis technique proposed by Menon et al. [15]. In that system, the application state is represented as a state transition table, where each row denotes occupant o_i and each column indicates zone z_j as shown in Figure 6. In their system, an *event* (generated by the event generation algorithm) is a set of person-probability pairs, $< o_i;\ p(o_i) >$, where $p(o_i)$ is the probability that the detected signature matched occupant o_i. Using the event vector as defined above, the algorithm computes the new state transition table by applying the event vector to the current state. In the zone in which the event occurred, the new probability for an occupant, $p_s(o_i)$ is calculated as follows: $p_s(o_i) = p(o_i) + x_i * p'_s(o_i)$, where $x_i = 1 - p(o_i)$ and $p'_s(o_i)$ is the probability of the occupant in the previous state. For all other zones, $p_s(o_i) = x_i * p'_s(o_i)$. This ensures that the sum of probabilities for an occupant across all zones in the new state transition table equals 1.

IV. Evaluation

The system architecture presented in Section II shows qualitatively how a domain expert can rapidly prototype a large-scale instance of a spatio-temporal analysis engine on a large-scale camera network by providing a set of handlers to our distributed system. In this section, we turn our attention to a quantitative study of the two design choices and how they may be used by a domain expert. Specifically, we would like to be able to give guidance to the application designer on the quantitative implications of the design choices relative to the application requirements.

In practice, spatio-temporal analysis applications would have a highly dynamic workload for event generation depending on various factors including number of cameras, number of known signatures, number of moving occupants, degree of mobility of the occupants and so on. In order to implement a real-time spatio-temporal analysis on a large-scale smart camera network, a domain expert needs guidance on the right amount of system resources needed to satisfy application requirements expressed as a combination of quality of service and estimated workload. We define three parameters for estimating the workload:

Signature Generation Rate
: Signature generation rate is defined as the number of signatures generated across all smart cameras per unit time. This parameter helps to abstract the scale of the application, which is really a combination of the number of cameras and the degree of mobility of the occupants.

Number of Signature Comparisons per Event
: This is the number of known signature comparisons needed for a single event generation. If there are more signatures in the database, it takes longer to generate an event since there are more signature comparisons involved. When spatio-temporal filtering is used, the number of actual signature comparisons is dynamic. However, a domain expert should specify an expected number of signature comparisons per event in order to estimate resource requirement for the system.

Signature Comparison Algorithm
: Different signature comparison algorithms have different complexities and execution times. A signature comparison algorithm should be specified in order to model the workload of event generation.

We also define two parameters that would help specify the quality of service constraints for the application.

Event Generation Latency
: Event generation latency specifies an upper bound for the time taken for a single event generation, i.e., the time between signature arrival and event generation. Application developer can specify a desired event generation latency as an application requirement.

Number of Missed Deadlines
: This parameter expresses another quality of service metric of the application, namely, the tolerance to missed deadlines. It is expressed as the number of events per unit time that can have longer latencies

[2]Please note that our intent in implementing a mockup of a spatio-temporal analysis engine on top of our distributed system is to mimic the workload that would be seen by a signature comparison server in an actual system for purposes of performance evaluation to be discussed in the next section.

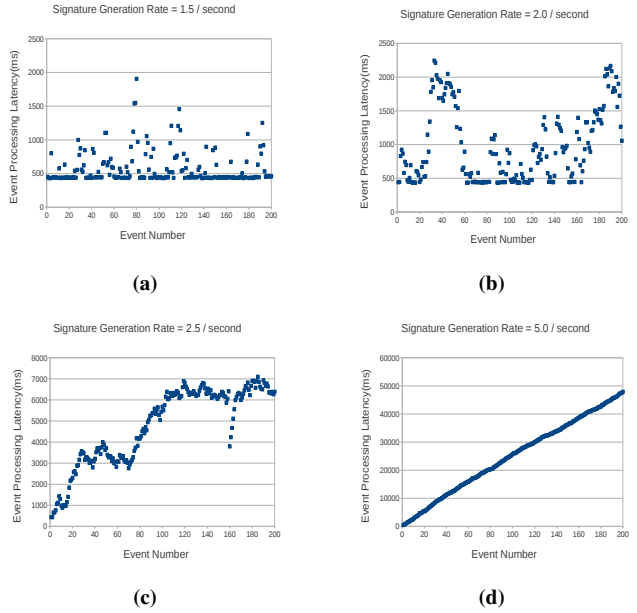

Fig. 7. Centralized event generation: Event processing latency with centralized processing for event generation with different signature generation rate: (a) 1.5 (b) 2.0 (c) 2.5 (d) 5.0 per second

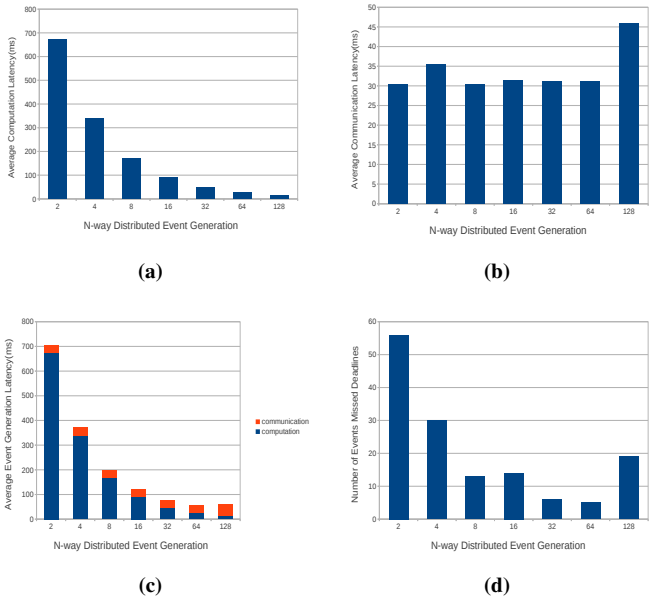

Fig. 8. Distributed event generation: (a) average computation latency (b) average communication latency (c) average event generation latency (sum of computation and communication) (d) number of missed deadlines

than the specified event generation latency.

Based on given parameters, we conduct a set of experiments to show how a developer can make decisions on system design and the amount of system resources needed to satisfy application quality of service expectations. We also show quantitative measurements showing how the design decisions affect application behavior under various conditions.

To mimic a realistic physical environment, we add Gaussian random noise to intervals between signatures dictated by signature generation rate given as a parameter. This will help create bursty signature generation and stress test the system. To provide the signature generation rate as a parameter, we use a signature generator as the workload generator instead of real distributed smart cameras running actual video analytics. The signature generator takes a application-specified signature generation rate as the input and generates signatures with intervals plus random noises.

Scalability of Centralized Event Generation: Based on the defined parameters, we conduct an experiment showing the scalability of centralized event generation. In this experiment, We compared 200 signatures against each generated signature using Bayesian face recognition algorithm.

Figure 7 shows the event generation latency of each event using the centralized event generation. Each experiment is conducted with different signature generation rates to show various workload condition. To define different workload conditions, we first measure the maximum number of events that can be generated from the centralized system by giving a very high signature generation rate (100 per second). Based on this observation, we determined that the centralized system can handle up to 2 signatures per second. Figure 7(a) shows a scatter plot of the latency incurred by each event in underloaded condition. As shown in the figure, most of the events are generated within 500 milliseconds after signatures arrive. There are some events generated with long latencies however, due to burstiness in signature arrival. Figure 7(b) shows a situation when the system resources are fully utilized. Although event generation latencies are higher due to the higher chance of bursty situations, the system is able to handle all the signatures that arrive. Figure 7(c) depicts an overloaded situation. In this case, more signatures than the centralized system capacity arrive, and latencies for each event gradually grows. In Figure 7(d), the system is completely overloaded and the latency of each event linearly increases because there are always other signatures in the queue when a new signature arrives.

These experiments show the limitations of centralized event generation in two different ways. First, centralized event generation is not scalable in terms of event generation throughput, i.e., it has a bounded capacity for event generation. This may not be a viable option for a scalable spatio-temporal analysis engine that would like to react to the application dynamics and increase the throughput for event generation when the signature generation rate goes up. As can be seen from the results, if the signature generation rate exceeds the centralized system capacity then the system will not be able to meet the event generation latency quality of service metric. Secondly, the centralized system is incapable of handling burstiness in signature arrival even if the arrival rate is within the centralized system capacity. Once again the upshot of burstiness in signature arrival is longer latencies for some event generation, which may violate the number of missed deadlines quality of service metric of the application.

Resource Use of Distributed Event Generation: Since the signature comparisons are embarrassingly parallel, distributed event generation is scalable by design if unlimited number of

nodes can be harnessed to meet the application dynamics both in terms of signature generation rate and burstiness. Therefore, we do a more useful evaluation that would be prescriptive to the application designer, namely, the right amount of system resources (i.e., number of nodes) that is needed to meet the application quality of service metrics in terms of event generation latency, and number of missed deadlines.

When an application has a specific requirement for event generation latency, a system designer should find an economic choice on the number of nodes deployed while meeting the requirement. To reduce event generation latency, signature comparisons for a single event generation should be distributed over multiple nodes. For example, if there are N worker nodes available, we can assign $\frac{X}{N}$ comparisons for each node where X is the total number of comparisons for a single event generation. If the number of worker nodes increases, the latency of event generation reduces due to the exploitation of parallelism using the distributed nodes.

For the distributed experiments, we measured average latencies of 300 events while each event involves comparing a face feature to 200 face features using Bayesian face recognition algorithm. For this set of experiments, we fixed the signature generation rate fixed at 2 signatures per second. This choice ensures that there will be no missed deadlines even with the smallest degree of parallelism used in the experiment (i.e, since we determined that 2 signatures per second is the capacity of 2 EC2 nodes, the minimum degree of parallelism that we used in our experiments),

In Figure 11, we show the latencies experienced for event generation as a function of the number of nodes deployed for performing the signature comparison. Figure 8(a) shows the computation latency for event generation, which essentially captures the time it takes to compare signatures. As the number of nodes exponentially increases, the computation latencies exponentially decreases since workload of each node linearly decreases with the number of nodes. This is as expected since the computation for signature comparison is embarrassingly parallel Figure 8(b) shows the communication cost including transmitting a signature to distributed worker nodes and gathering the comparison results. The communication cost is quite stable, although there is a dramatic increase with 128 nodes. We found that the variance in communication latencies is quite large with 128 nodes, which is the reason for the increased average communication latency with 128 nodes. It is also noteworthy that the communication time dominates the computation time when 128 nodes are deployed. In other words, there is no benefit in deploying more than 64 nodes for this problem size (in terms of signature comparison workload).

Figure 8(c) shows overall event generation latency including the computation and communication latency. Figure 8(c) can be used in a prescriptive manner by the domain expert to find a cost-effective (i.e., in the number of nodes that need to deployed) solution to meet the application's quality of service needs. For example, if application wants to have less than 200ms delay for each event generation, it may choose 8 nodes for signature comparison since it is the minimum number of

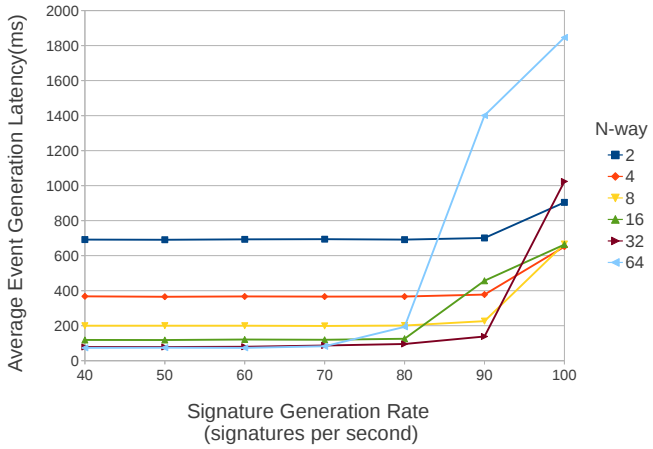

Fig. 9. Sensitivity of distributed event generation to signature generation rate

nodes meeting the requirement.

However, due to the dynamic nature of physical environments, there can be unexpected longer latencies when signatures are generated at the same time. Figure 8(d) shows the number of events that missed their deadlines. We set the deadline as 1.2 times of average latency of different settings acquired from Figure 8(c) by assuming an application scenario where the application can tolerate 120% of expected event generation latency. As shown in Figure 8(d), increasing the degree of parallelism for event generation is beneficial to reducing the number of missed deadlines. Once again we notice that the missed deadlines goes up for 128 nodes. As we observed earlier, the communication time starts to dominate computation time past 64 nodes. This coupled with the huge variance we observed for communication latency with 128 nodes leads to increased missed deadlines. Thus while in general increasing the number of signature comparison nodes to meet the two quality of service requirements of average event latency and missed deadlines is a good idea, there is a point of diminishing returns when the communication time exceeds the computation time.

The next experiment is designed to show the choice that an application developer may exercise between latency reduction and workload fluctuations with fixed set of resources. We use a fixed set of 128 nodes in this experiment. Figure 9 shows the event generation latency as a function of signature generation rate for different degrees of parallelism used to handle each event generation. For a given degree of parallelism employed, the average event generation latency remains pretty much unchanged as the workload increases until the system gets saturated due to unavailability of resources to handle incoming new requests. The system saturation point is reached earlier when more nodes are deployed for each event generation. For example, when only 2 nodes are used per event generation the system saturation point is reach at about 90 signatures per second. On the other hand, the system saturation is reached much earlier (around 70 signatures per second) when 64 nodes

are used per event generation. Also, the performance degrades more gracefully when less nodes are used per event generation (notice the slopes of the lines beyond the system saturation point) with increasing workload.

V. Related Work

There are many distributed systems that support smart environments by acquiring and applying knowledge about the environment and its inhabitants [16]. For example, the EasyLiving project [17] presents a software architecture and for smart environments that includes a person tracking system based on color stereo for maintaining the identities and locations of people. Gaia meta-operating system [18] provides a framework for building user-centric applications in active spaces, where users seamlessly interact with their surrounding physical and digital environments. SLIPstream [19] presents a scalable programming framework that allows interactive perception applications to run on distributed nodes to process streaming data with low latency. The focus of such systems is room or building scale smart environments; our focus is large-scale camera networks and the application focus for us is real-time spatio-temporal analysis, quite different from these other systems.

There have been extensive research in the area of individual identification using biometric information. For example, there are existing systems that can accurately recognize an identity by comparing one face to thousands of faces in a database [20]. BioID [21] uses multi-modal biometric information such as face, lip movement, and voice for person identification in order to achieve higher accuracy than using a single biometric information. Tulyakov et al. [5] show verification and open set identification systems that require different matchers for their best performance. These advanced biometric identification methods can be used in our system in order to improve accuracy of signature generation and comparison.

VI. Conclusion

Distributed smart camera networks are being widely deployed from building scale to city scale. Applications for such camera networks are wide ranging and encompass surveillance, transportation, assisted living, and the like. Video analytics and spatio-temporal reasoning are techniques used for converting raw data streams from these cameras to actionable knowledge. There are serious impediments to scaling such techniques to large-scale camera networks.

In this paper, we have identified the design choices in constructing a real-time spatio-temporal analysis engine using a large-scale camera network. We present a distributed system architecture that would enable domain experts to rapidly prototype their applications with minimal effort. We have implemented our system using Amazon EC2 for distributing the compute-intensive parts and have conducted performance evaluation that would serve a domain expert to make intelligent design decisions to meet the quality of service needs of the applications.

VII. Acknowledgment

Our sincere thanks to Amazon for generously providing the computational resources on EC2 for carrying out our experiments. Our thanks are due to our colleagues in the embedded pervasive lab, particularly Dave Lillethun.

References

[1] "Schiphol Airport leverages technology to drive efficiency." [Online]. Available: "http://www.securitysystemsnewseurope.com/?p=article&id=se200906VF2Isw"

[2] U. Ramachandran, K. Hong, L. Iftode, R. Jain, R. Kumar, K. Rothermel, J. Shin, and R. Sivakumar, "Large-scale situation awareness with camera networks and multimodal sensing," *Proceedings of the IEEE, (Centennial year special issue on the Frontiers of Audiovisual Communications: New Convergences of Broadband Communications, Computing and Rich Media)*, 2012(To Appear).

[3] V. Menon, B. Jayaraman, and V. Govindaraju, "The Three Rs of Cyberphysical Spaces," *Computer*, vol. 44, pp. 73–79, 2011.

[4] J. Zhang, Y. Yan, and M. Lades, "Face recognition: eigenface, elastic matching, and neural nets," *Proceedings of the IEEE*, vol. 85, no. 9, pp. 1423 –1435, sep 1997.

[5] S. Tulyakov, C. Wu, and V. Govindaraju, "On the difference between optimal combination functions for verification and identification systems," *IJPRAI*, vol. 24, no. 2, pp. 173–191, 2010.

[6] S. Tulyakov and V. Govindaraju, "Issues and advances in biometrics," 2009.

[7] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, pp. 558–565, July 1978.

[8] K. Hong, S. Smaldone, J. Shin, D. J. Lillethun, L. Iftode, and U. Ramachandran, "Target container: A target-centric parallel programming abstraction for video-based surveillance," in *ICDSC*, 2011, pp. 1–8.

[9] "Apache ActiveMQ." [Online]. Available: "http://activemq.apache.org/"

[10] "Amazon Elastic Compute Cloud." [Online]. Available: "http://aws.amazon.com/ec2/"

[11] "OpenCV Wiki," 2010. [Online]. Available: http://opencv.willowgarage.com/wiki/

[12] D. Comaniciu, V. Ramesh, and P. Meer, "Real-time tracking of non-rigid objects using mean shift," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 2, p. 2142, 2000.

[13] "CSU Face Identification Evaluation System website," 2010. [Online]. Available: http://www.cs.colostate.edu/evalfacerec/algorithms5.php

[14] J. P. Phillips, H. Moon, S. A. Rizvi, and P. J. Rauss, "The FERET Evaluation Methodology for Face-Recognition Algorithms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 10, pp. 1090–1104, 2000.

[15] V. Menon, B. Jayaraman, and V. Govindaraju, "Multimodal identification and tracking in smart environments," *Personal Ubiquitous Comput.*, vol. 14, pp. 685–694, December 2010. [Online]. Available: http://dx.doi.org/10.1007/s00779-010-0288-6

[16] D. J. Cook and S. K. Das, "How smart are our environments? An updated look at the state of the art," *Pervasive Mob. Comput.*, vol. 3, pp. 53–73, March 2007.

[17] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer, "EasyLiving: Technologies for Intelligent Environments," in *Handheld and Ubiquitous Computing*, ser. Lecture Notes in Computer Science, P. Thomas and H.-W. Gellersen, Eds. Springer Berlin / Heidelberg, 2000, vol. 1927, pp. 97–119.

[18] M. Roman, C. Hess, R. Cerqueira, A. Ranganathan, R. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *Pervasive Computing, IEEE*, vol. 1, no. 4, pp. 74 – 83, oct-dec 2002.

[19] P. S. Pillai, L. B. Mummert, S. W. Schlosser, R. Sukthankar, and C. J. Helfrich, "Slipstream: scalable low-latency interactive perception on streaming data," in *Proceedings of the 18th international workshop on Network and operating systems support for digital audio and video*, ser. NOSSDAV '09. New York, NY, USA: ACM, 2009, pp. 43–48.

[20] A. Pentland and T. Choudhury, "Face recognition for smart environments," *Computer*, vol. 33, no. 2, pp. 50 –55, feb 2000.

[21] R. Frischholz and U. Dieckmann, "Biold: a multimodal biometric identification system," *Computer*, vol. 33, no. 2, pp. 64 –68, feb 2000.